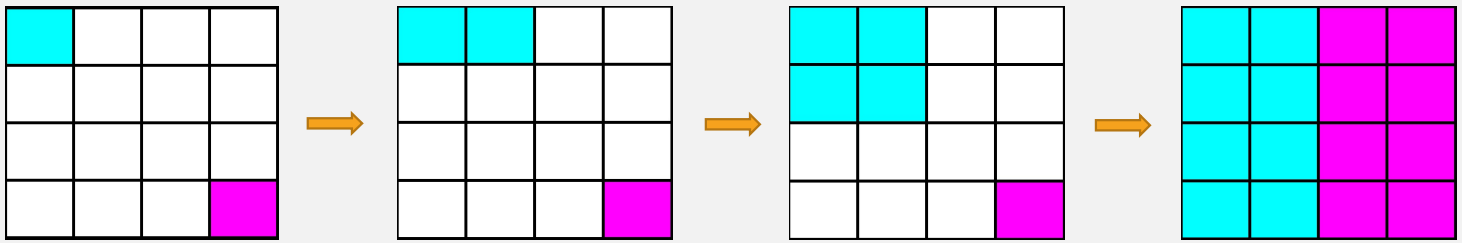
MÉTODOS DE PESQUISA HEURÍSTICA PARA RESOLUÇÃO DE PROBLEMAS

JOGO: “FOLDING BLOCKS”

INTRODUÇÃO

- Neste projeto é pretendido implementar diversos algoritmos de pesquisa capazes de vencer uma série de níveis do jogo "**Folding Blocks"**.

- Este jogo, inicialmente composto por um tabuleiro com uma ou mais peças, tem como objetivo, ocupar todos os espaços dos tabuleiros utilizando essas peças.



FORMULAÇÃO DO PROBLEMA

- **Representação do estado**

O tabuleiro do jogo é representado por uma matriz com tamanho variável (de acordo com o nível). Esta matriz é do tipo int[ ][ ], cujo valores podem ser:

  - 0 - caso de espaços vazios;
  - N - sendo 0 < N < Tamanho máximo do tabuleiro, onde a cada N corresponde um tipo de peça.

- **Teste Objetivo**

O jogo acaba quando o tabuleiro não tiver espaços vazios. Ou seja, quando em todas as posições do tabuleiro x e y, board[x][y], seja diferente de 0.

FORMULAÇÃO DO PROBLEMA

Nomes	Pré-condições	Efeitos	Custos
Dobrar para a Esquerda	Yb >= 0; Tab[xi][yb] = 0;	Dist = (yi - y_eixo) + 1; Yb = y_eixo - Dist; Tab[Xi][Yb] = ID;	1
Dobrar para a Direita	Yb < M; Tab[xi][yb] = 0;	Dist = (y_eixo - yi) + 1; Yb = y_eixo + Dist; Tab[Xi][Yb] = ID;	1
Dobrar para Cima	Xb >= 0; Tab[xb][yi] = 0;	Dist = (xi - x_eixo) + 1; Xb = x_eixo - Dist; Tab[Xb][Yi] = ID;	1
Dobrar para Baixo	Xb < N; Tab[xb][yi] = 0;	Dist = (x_eixo - xi) + 1; Xb = x_eixo + Dist; Tab[Xb][Yi] = ID;	1

- ID - referente à peça a ser jogada;
- Tab[ ][ ] - valor na posição de cada tabuleiro (pode ter os seguintes valores: 0 - se estiver livre / ID - numero referente à peça a ser jogada);
- Xi / Yi - coordenada referente à linha/coluna (respetivamente) do bloco atual;
- Xb / Yb - coordenada referente à linha/coluna (respetivamente) do bloco novo;
- x_eixo / y_eixo - coordenada referente à linha/coluna (respetivamente) do eixo de simetria;
- Dist - distância ao eixo de simetria;
- N x M - dimensões do tabuleiro de jogo (linhas/colunas);

IMPLEMENTAÇÃO DO JOGO

- Linguagem escolhida para desenvolvimento do código: **Java;**
- Trabalho realizado em **Eclipse** & **VSCode;**
- Código foi dividido pelas seguintes classes:
  - Main.java
  - Game.java
  - Level.java
  - Logic.java
  - Algoritmo.java

ALGORITMOS E HEURÍSTICAS

- **“Depth First Search” ou Pesquisa em Profundidade:** é um algoritmo que tem como estratégia expandir sempre primeiro o nó de maior profundidade.

  - dada a existência de várias soluções e a irrelevância da sequência em que as peças são jogadas mostra-se um algoritmo rápido e eficiente em níveis de menor dificuldade;

  - Para casos de níveis mais complexos em que existe um maior número de jogadas que podem levar a resoluções impossíveis, o método de pesquisa prova-se ineficiente pois explora demasiados nós irrelevantes devido a falta de critério na seleção destes mesmos.

ALGORITMOS E HEURÍSTICAS

- **"Greedy" ou Pesquisa Gulosa:** este algoritmo tem como estratégia expandir o nó que está mais perto da solução. Utiliza uma função heurística que devolve um custo estimado do caminho mais curto do estado n para o objetivo.

$$f(n) = heurística(n)$$

- Com este algoritmo foram utilizadas duas heurísticas diferentes:
  - Heurística2:

$$heurística2(n) = número\ de\ espaços\ preenchidos$$

  - Heurística3:

$$heurística3 = log_2\left(\frac{capacidade\ do\ tabuleiro}{espaços\ preenchidos}\right) * número\ de\ peças$$

ALGORITMOS E HEURÍSTICAS

- **"A*":** o algoritmo A* combina a pesquisa gulosa com a de custo uniforme minimizando a soma do caminho já efetuado, com o mínimo previsto/estimado que falta até à solução. A solução do algoritmo A* é ótima é completa sendo importante o uso de uma heurística adequada.

$$f(n) = heurística(n) + custo(n)$$

- A heurística utilizada foi a seguinte:
  - Heurística4:

$$heurística4(n) = log_2\left(\frac{capacidade\ do\ tabuleiro}{espaços\ preenchidos}\right)$$

ALGORITMOS E HEURÍSTICAS

- **Implementação própria:** Neste caso o algoritmo atribui a cada nó o valor de acordo com a heurística em baixo representada e dá prioridade aquele com maior valor. É dada prioridade aos nós de maior valor.
  - Heurística5:

$$heurística5(n) = (tamanho\ máximo\ da\ peça\ jogada) + (número\ de\ espaços\ preenchidos) - (custo)$$

COMPARAÇÃO DE RESULTADOS

- Por norma o valor de nós utilizados vai estar diretamente relacionado com tempo, dado que quantos mais nós o algoritmo explora mais tempo gastará a encontrar a solução. Podemos reparar que em geral a pesquisa para os algoritmos Greedy e DFS são menos extensas e como consequência mais rápidas.

TIME (ms)

	DFS	Greedy H2	Greedy H3	A*	Our Own
Level 1	4	5	5	5	5
Level 2	4	5	5	9	5
Level 3	5	5	7	14	5
Level 4	5	5	6	7	5
Level 5	5	6	7	113	9
Level 6	5	7	29	95	7
Level 7	5	7	11	296	8
Level 8	7	10	9	971	8
Level 9	6	8	16	ND	715
Level 10	ND	ND	ND	ND	13284
Level 11	11	15	11366	ND	288

Number Of Nodes

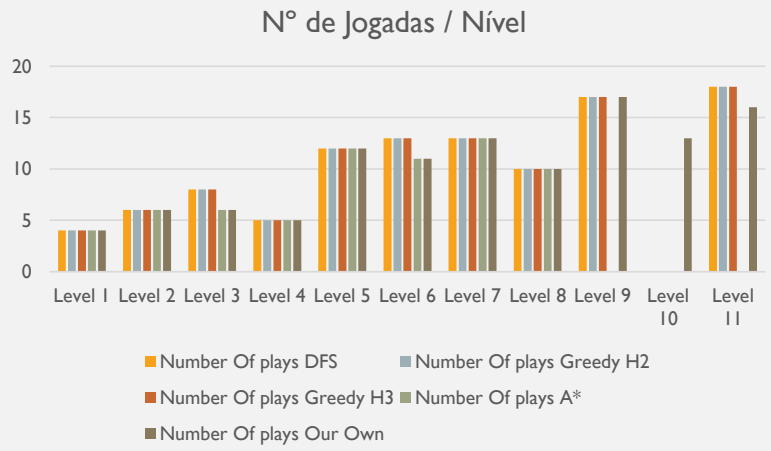
	DFS	Greedy H2	Greedy H3	A*	Our Own
Level 1	4	4	4	11	4
Level 2	6	6	9	60	6
Level 3	8	8	11	140	7
Level 4	5	5	7	34	5
Level 5	12	12	26	2374	26
Level 6	13	13	444	1931	13
Level 7	13	13	48	6651	27
Level 8	58	70	36	34871	25
Level 9	17	17	55	ND	22235
Level 10	ND	ND	ND	ND	191171
Level 11	97	85	217690	ND	5907

COMPARAÇÃO DE RESULTADOS

- O número de jogadas utilizadas vai depender do nível em questão e do algoritmo utilizado. O A* vai oferecer sempre a melhor jogada perante o nível que lhe é apresentado, no entanto, acontece que em alguns níveis o número de jogadas até à solução é fixo, mesmo havendo várias soluções.

Number Of plays

	DFS	Greedy H2	Greedy H3	A*	Our Own
Level 1	4	4	4	4	4
Level 2	6	6	6	6	6
Level 3	8	8	8	6	6
Level 4	5	5	5	5	5
Level 5	12	12	12	12	12
Level 6	13	13	13	11	11
Level 7	13	13	13	13	13
Level 8	10	10	10	10	10
Level 9	17	17	17	ND	17
Level 10	ND	ND	ND	ND	13
Level 11	18	18	18	ND	16



CONCLUSÕES

- A escolha do algoritmo de pesquisa utilizado tem de ser ponderada e avaliada perante o parâmetro que pretendemos otimizar.

- Se o interesse do utilizador for rapidez, então o algoritmo DFS ou Greedy será uma boa escolha e até certo nível escalável, sendo que para níveis onde os tabuleiros de jogo são bem maiores mantém na mesma um tempo de resposta rápido.

- Se o interesse residir na capacidade de terminar o jogo no menor número de jogadas possíveis, então, nesse caso A* apresenta-se como uma boa opção, garantindo sempre o menor número de jogadas.

- O algoritmo por nós implementado apresentou resultado interessantes, tendo sido o único a conseguir completar todos os níveis em tempos definidos e com rapidez, garantindo sempre a solução ótima.

REFERÊNCIAS

- GeeksforGeeks:
  - https://www.geeksforgeeks.org/greedy-algorithms/
  - https://www.geeksforgeeks.org/search-algorithms-in-ai/
- Red Blob Games:
  - https://www.redblobgames.com/pathfinding/a-star/introduction.html?fbclid=IwAR0_pdUYGGfwkUHWjwAkKdxsq_A-1HqhF3XDIaMC0Wx95q_WESjoRqyklUU
  - http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
  - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm
- Slides da Matéria lecionada em aula.